Object-Oriented Design (OOD)

OOA dealt with conceptual/domain models – OOD is concerned with the design of a software system.

The goal of OOD is to develop a specification class diagram (as opposed to a conceptual class diagram).

A specification class diagram describes the actual classes and interfaces of a software system (instead of concepts of a problem domain).

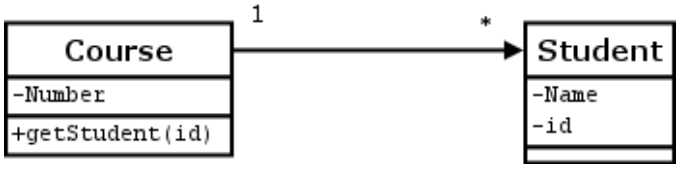
In a specification class diagram class associations are interpreted as responsibilities. Class associations interpreted as responsibilities are unidirectional. One way to view responsibilities is that one class has to keep track of the other class.

Example:

Here the course class has the responsibility to keep track of all the students that attend it.

In specification class diagrams the classes are enriched by adding methods and visibility constraints (+ ≅ public, - ≅ private).

Example:



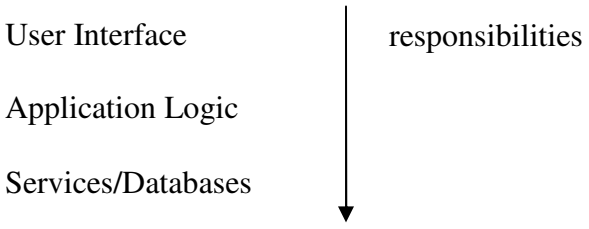
Here, given a student id the course object can return the appropriate student object.

High-level Guidelines for OOD

i. Identify classes taking inspiration from the conceptual class models
   - not everything from the domain models has to appear in specification diagrams

ii. Look for standard design patterns.

iii. Partition responsibilities among classes by using encapsulation principles – "put methods with the data."
iv. Judge the design by how well it can withstand change, usually based on low coupling (inter object) and high cohesion (intra object):
    - Coupling is the degree to which one module depends on another.
    - Cohesion is the degree to which the parts of a module depend on each other.
v. Consider a 3-layered architecture:



In a 3-tier architecture the responsibilities always point from the upper layers to the lower layer, e.g., a DB never keeps track of user interfaces, but user interfaces will keep track of DBs for display purposes and goal oriented computation.

☞ Lower layers should never depend on higher layers.

vi. Reuse classes whenever possible.
vii. A class should capture exactly one key abstraction.
viii. Keep related data and behavior in one class (high cohesion).
ix. Distribute system intelligence among the classes.

Ultimately, Behavior Modeling → behavior of system needs to fulfill requirement set forth in the use case texts.